前缀

2018年12月12日 19:15

- 前缀

一. 前缀和prefix

1. 前i项和S[i]=ΣA[x]（x从1到i）
2. l到r项和sum(l,r)=S[r]-S[l-1]
3. 二维前缀和S[i][j]=ΣΣA[x][y]（x从1到i，y从1到j）
   i. 或S[i][j]=S[i-1][j]+S[i][j-1]+A[i][j]-S[i-1][j-1]
4. 二维矩形和=S[I][J]+S[I-i][J-j]-S[I-i][J]-S[I][J-j]
   i. 其实是容斥原理的思想
5. 如果要给[l,r]区间每个数++，只需++a[l]，--a[r+1]，最后对a求前缀和

二. 前缀和例题

1. 求n种面值的硬币各一块时，最小的不能凑成的钱
   i. 没有1时，1即为答案
   ii. 有1时，根据前缀和是递增的，循环到第一个比前缀和大的数为止，前缀和+1即为答案
   iii.
```
sort(a,a+n);
if(a[0]!=1){
        cout<<1;
        return 0;}
s[0]=a[0];
for_(i,1,n)
        s[i]=s[i-1]+a[i];
for_(i,1,n)
        if(s[i-1]+1<a[i]){ //若第n+1项-前n项和>1，前n项和+1则为答案
                cout<<sum+1;
                return 0;}
cout<<s[n]+1;
```
2. 求n种面值的硬币各无限块时，想凑出1~m任意值需要的最少硬币数
   i. 思路同上，不过这次只有不存在1时才会有凑不出的数了，否则大于前缀和的时候，疯狂增加前一项的数量即可，而且这样一定数量最小
   ii. 注意有时前缀和比本身大了，此时可能会算出负数，所以不能用ull
   iii.
```
sort(v,v+n);
if(v[0]!=1){
        cout<<-1;          //凑不出1
        return 0;}
v[n]=m;//便于判断最大一种硬币也远小于m的情况
ll s=0;
for_(i,0,n){
        if(v[i+1]>=m){
                ll t=(m-s)/v[i];
                if(t*v[i]+s<m)
                        ++t;     //手动向上取整，顺便避免负数情况
                ans+=t;
```

```
            break;}
        ll t=(v[i+1]-s-1)/v[i];
        if(t*v[i]+s<v[i+1]-1)
            ++t;    //手动向上取整，顺便避免负数情况
        s+=t*v[i];
        ans+=t;}
    cout<<ans;
```

3. 平面直角坐标系中，边平行于轴的矩形所能围出的，边长<=10的最大和

```
for__(i,1,n)
    for__(j,1,m)
        scanf("%d",s[i]+j);
for__(i,1,n)
    for__(j,1,m)
        s[i][j]+=s[i-1][j]+s[i][j-1]-s[i-1][j-1];
for__(I,1,n)
    for__(J,1,m)
        for_(i,max(I-10,0),I)
            for_(j,max(J-10,0),J)
                ans=max(ans,s[I][J]+s[i][j]-s[I][j]-s[i][J]);
```

4. Tallest Cow
   i. 分析：若I可以看见r，--(l,r)区间的a；已知最高牛p的高度为H，其a应为0，其他牛的a都<=0，则H+S[i]即为可行的i牛高度；m个看见关系中可能存在重复输入，可以用序偶到布尔值的映射关系判断，为了方便判断重复，在制作序偶前先ifswap
   ii. 时间：通过前缀和，从O（MN）优化到O（M+N）
   iii.

```
int main() {
    map<pair<int,int>,bool>existed;
    scanf("%d%d%d%d",&n,&p,&h,&m);
    int l,r;
    for_(i,0,m){
        scanf("%d%d",&l,&r);
        if(l>r)   //确保是从左往右看的顺序
            swap(l,r);
        if(existed[make_pair(l,r)])
            continue;        //防重复输入
        existed[make_pair(l,r)]=true;
        --a[l+1];
        ++a[r];}//给（l，r）区间-1，即给[l+1,r)区间-1
    for__(i,1,n){
        S[i]=S[i-1]+a[i]; //前缀和
        printf("%d\n",S[i]+h); }
    return 0;}
```

5. 2维三角区域钓鱼
   i. 数据：第一象限，m个渔夫，可钓到斜边为x轴，长2l，高l的等腰三角形的鱼，所有输入数据>0，<200000

ii. 分析：每读入一条鱼的位置，就反向分析一波哪个区间能钓到它，用前缀和优化时间

iii.
```
for_(i,0,n){
        scanf("%d%d",&x,&y);
        if(y<=l){//否则谁都钓不到，不管
                ++t[max(0,x-l+y)];
                --t[1+min(200000,x+l-y)];}}
ans[0]=t[0];
for__(i,1,200000)
        ans[i]=ans[i-1]+t[i];
for_(j,0,m){
        scanf("%d",&x);
        printf("%d\n",ans[x]);}
```

三. 差分例题

1. 通过区间+1或-1使数组变成同一数的最小操作次数，及该次数下的变成同一数的可能取值数量（USSTD7C）

i. 将区间修改映射到对差分的区间变化

ii. 易知将正的和负的都变成0以后即为答案

iii. 但a0~a1的差分和an~an+1的差分就只用于微调

```
int a[MN],d[MN];//d[i] = a[i]-a[i-1]
ll z,f;                 //正的差分的总和，负的差分的绝对值的总和
//max(z,f)为距离a[1]最远的差值，abs(z-f)为abs(a[1]-a[n])
```

iv.
```
for__(i,2,n)
        if(d[i]>0)
                z+=d[i];
        else
                f-=d[i];
printf("%lld\n%lld",max(z,f),abs(z-f)+1); //或max(z,f),abs(a[1]-a[n])+1
```